DOCUMENT RESUME

ED 097 918 IR 001 298

AUTHOR Danielson, Ronald L.; Nievergelt, Jurg
TITLE An Automatic Tutor for Introductory Programming Students.
INSTITUTION Illinois Univ., Urbana. Dept. of Computer Science.
SPONS AGENCY National Science Foundation, Washington, D.C.
PUB DATE 74
NOTE 9p.

EDRS PRICE MF-$0.75 HC-$1.50 PLUS POSTAGE
DESCRIPTORS *Computer Assisted Instruction; *Computer Programs; Computers; *Computer Science Education; Individualized Instruction; *Problem Solving; *Programed Tutoring; Tutorial Programs
IDENTIFIERS *PLATO IV; University of Illinois

ABSTRACT

A program was developed to use the PLATO IV system of the University of Illinois to help students solve typical programing problems. The program tries to approximate a near-ideal situation in which each student receives correction of logical errors and comments on good programing practice as he goes along in a one-on-one tutorial environment. The tutor program utilizes an AND-OR graph as a representation of all reasonably correct approaches to the particular problem, as well as many of the wrong approaches introductory students are likely to attempt. The computer-assisted instruction program gives students the personal attention they need for learning the problem solving of computer programs. (WH)

ED 037918

An Automatic Tutor for Introductory Programming Students*

Ronald L. Danielson
Jurg Nievergelt

Department of Computer Science
University of Illinois
Urbana, Illinois

U S DEPARTMENT OF HEALTH EDUCATION & WELFARE NATIONAL INSTITUTE OF EDUCATION
THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

Introduction

Beginning with the work of Dijkstra [2], there has been a growing interest in the computing field in the use of proper program structure in the solution of programming problems. Considerations of ease of debugging and program maintainability place increasing importance on teaching good program structure as an aid to problem solution.

Unfortunately, as Gries[6] has noted, while programming is essentially a problem solving activity, introductory programming courses typically concern themselves with the syntax of a particular programming language, and a few more or less relevant applications, but say nothing about how to solve a problem in general.

This is due to a number of factors, such as the problems of simply getting students to write correct solutions, as well as the difficulty of presenting the concepts involved in structured programming in a large lecture environment. A near-ideal situation might be for each student to construct his solutions in a one-on-one tutorial environment, enabling him to receive correction of logical errors and comments on good programming practice as he goes along. There is just not enough individual attention of this sort in the typical system of using teach-

* This work was supported in part by the National Science Foundation under Grant No. US NSF EC-41511.

IR001998

ing assistants and graders in conjunction with large lecture sections. Consequently, most students are never able to develop problem solving methods which result in well-structured solutions.

Previous attempts at developing machine tutors of programming [4,5,7,10] have not filled this need, primarily because of the simple problem examples they are concerned with and their inability to provide useful comments on the student's problem solving technique, as well as his particular solution.

This paper discusses an automatic tutor of programming, implemented on the PLATO IV CAI system [1] as a part of an automated instructional system for computer science being developed at the University of Illinois [8]. This tutor exposes the introductory student to structured programming concepts and top-down problem solution techniques by means of example. The student develops (on-line, using an approach based on stepwise refinement [11]) a solution to a single, relatively complex programming problem (symbolic differentiation of a polynomial). Student and tutor engage in an interactive dialogue during this problem-solving process, the tutor providing hints regarding solution steps, and comments about program structure and efficiency. A small sample of the types of responses the tutor might make is presented in Fig. 1.

Tutor's knowledge of solutions

Clearly, for the tutor to provide meaningful comments and assistance to all students, it must possess a detailed knowledge of the solution space for the particular problem with which it is concerned. Our tutor utilizes an AND-OR graph (Fig. 2 is a small portion) as a representation of all reasonable correct approaches to the particular problem, as well as many of the wrong approaches introductory students are likely to attempt.

An OR node represents a point in the development of a solution where there are several (equally correct) acceptable approaches, and the student may continue a correct solution by following any of the branches out of the node. An AND node, on the other hand, represents a point at which there are several tasks which must all be done in order for the solution to be correct, and the student must traverse all branches out of the node to achieve a correct solution.

Tutor: Tell me how you intend to proceed.
[The tutor's standard prompt]

Student: I don't know.

T: Try to think of a way to break the problem into simpler subproblems.
[A gentle hint for an AND node. There are several levels of hints for both types of nodes]

S: Look for any special cases.

T: OK. Let's call this task 1.

S: Then consider the general case.

T: OK. Let's call this task 2.
Now let's consider task 1.
Tell me how you intend to proceed.

S: I want to see if the derivative is zero.

T: OK. Let's call this task 1.1.
[A refinement at an AND node]

.
.
.

T: Tell me how you intend to proceed.
[See Fig. 2]

S: I'm going to loop through the term looking for what I'm differentiating by.
[This input matches an error branch]

T: Try using built-in functions instead.
[So the tutor displays the explicit hint, but the current node remains unchanged]

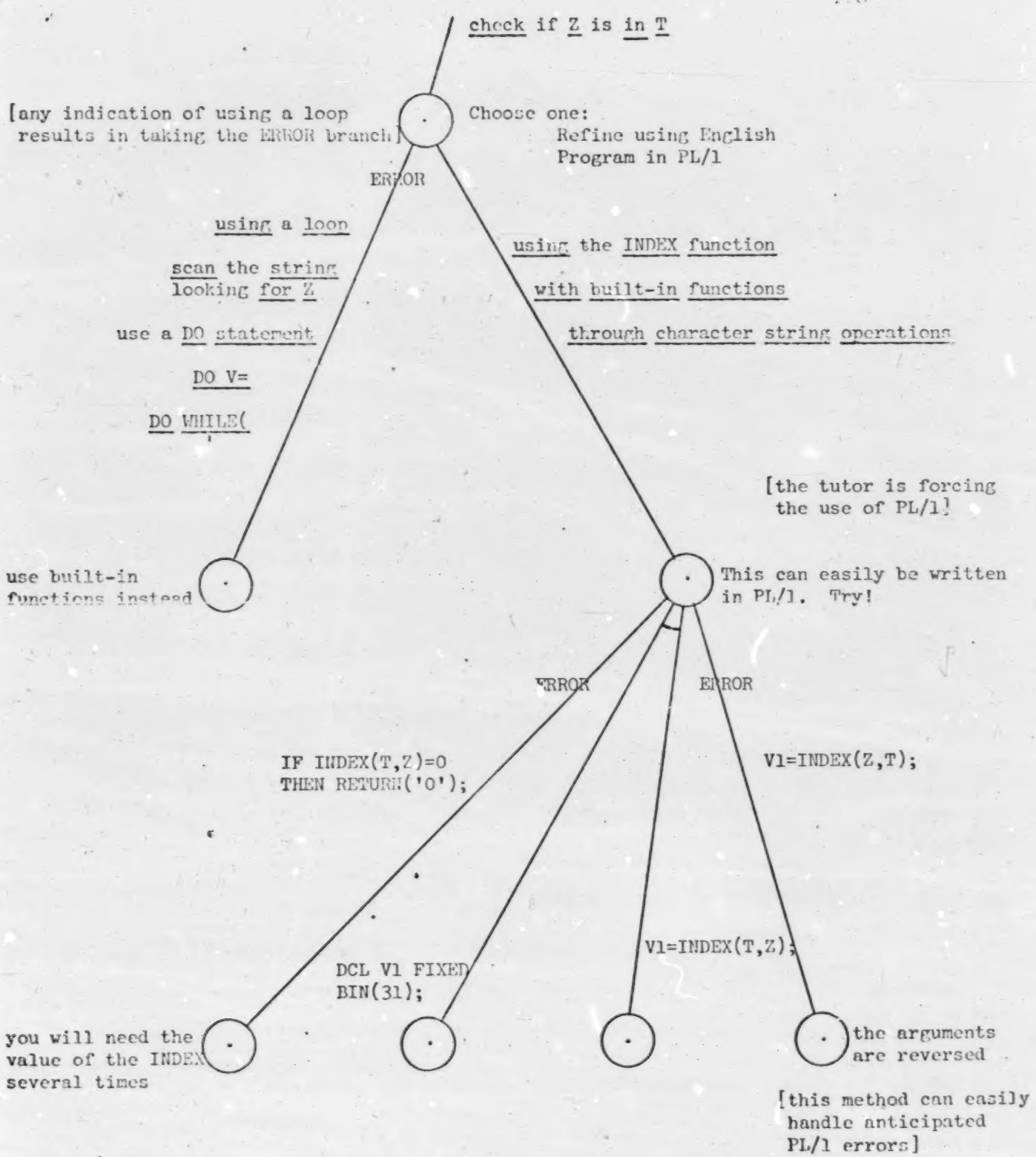
S: Maybe I want the INDEX function.

T: OK.

.
.
.

Author's comments in brackets []

Fig. 1


check if Z is in T
[any indication of using a loop results in taking the ERROR branch]
Choose one:
Refine using English
Program in PL/1
ERROR
using a loop
scan the string looking for Z
use a DO statement
DO V=
DO WHILE(
using the INDEX function
with built-in functions
through character string operations
[the tutor is forcing the use of PL/1]
use built-in functions instead
This can easily be written in PL/1. Try!
ERROR
ERROR
IF INDEX(T,Z)=0 THEN RETURN('0');
V1=INDEX(Z,T);
DCL V1 FIXED BIN(31);
V1=INDEX(T,Z);
you will need the value of the INDEX several times
the arguments are reversed
[this method can easily handle anticipated PL/1 errors]

Author's comments in brackets []

Fig. 2

Branches between nodes are associated with English phrases or programming language statements which the student must enter to traverse the graph. The next node in such a traversal is selected by the tutor based on the current node type and the student's inputs. Branches may be regular branches, traversed in response to correct steps in a problem solution, or error branches, which lead to detailed remedial comments which are presented to the student.

Note that such a graph represents all aspects of a problem solution. A path through the graph traces the refinement process the student went through in developing a solution, and the tip nodes, or more properly, the programming language statements associated with the branches leading to the tip nodes, represent the actual program which solves the problem. This representation is quite similar to the decision structure noted by Ells and Freeman [3] in a review of the design process of several large software projects.

The claim of being able to represent all reasonable solutions to a problem, as well as a few wrong approaches, as an AND-OR graph clearly requires some justification. Intuitively, one feels that the number of possible solutions must increase enormously as the complexity of the problem increases. Undoubtedly, this would be true if we were considering every correct solution; however, we are considering only "good" solutions, which makes an important difference. Many of the possible solutions (especially the solutions proposed by introductory students) may be immediately rejected on the basis of "goodness". Thus paths representing inefficient or poorly structured solutions need not be included in the graph, even though such solutions may be correct in a strict sense. At most, such solutions will be one of the plausible wrong approaches. Further, the use of a graph form, as opposed to a tree, allows maximum common utilization of sub-portions of the graph. These factors significantly reduce the number of nodes the solution graph must contain, and enable us to use such a representation.

Student-tutor interaction

The existence of a natural language, interactive dialogue between student and tutor is an essential part of our tutoring system. It is through this medium that the student indicates the refinements to be made in achieving a problem solution, and that the tutor provides hints and comments, and controls the traversal of the solution graph.

The PLATO IV author language, TUTOR, provides a facility for dialogue control which is essentially a simple keyword parsing scheme (see [9] for details). In general, such a scheme is not adequate for conducting a natural language discourse; however, the rigid local context provided by the current position in the solution graph (the only relevant inputs are associated with branches leaving the current node) limits the inputs the student might make, and allows our tutor to "understand" the input and match a branch in the solution graph a large percentage of the time.

It is unlikely, of course, that succesive student inputs will follow exactly the sequence expected by the solution graph. At any given node, some of the meaningful student inputs will match branches leaving that node, some will coalesce several levels of the solution graph into a single input, and some will match single branches at a greater depth into the graph. Inclusion of all possible student inputs in the solution graph would cause a tremendous increase in the number of branches leaving any node. Instead, we have chosen to include only the most likely student inputs, based on observation of students solving the problem, and the tutor conducts a depth-first search of the solution graph in the vicinity of the current node, attempting to match the student input to a branch further down the graph. Depth of the search is controlled by information on the average number of levels spanned by the student's previous inputs.

The tutor assigns "task names" to refinement steps as they are input by the student, based on the type of the current node in the solution graph. If the current node is an AND, several tasks will replace the old one, and an additional digit is added to the task name for each new task. That is,

1.2 figure total cost

becomes

1.2.1 subtract trade-in
1.2.2 add sales tax

On the other hand, if the current node is an OR, the new task simply replaces the old one, and the task name is unchanged. The student may preface each task name with an identifying letter if he so desires.

Task names are important for several reasons. Perhaps the most important is that they provide an easy and precise way for both the student and tutor to identify a particular task. Thus the tutor can display a "now refining

task ____" message so the student is always sure which task is being discussed, and the student can refer to task names in his input. This could be accomplished by simply using line numbers on the screen, but task names have the added advantage of indicating the history and relationship of statements, without making the underlying solution graph explicit. That is, statements F.2.3.1.2 and F.2.3.1.3 are brothers, having only recently been derived from a common father. On the other hand, tasks 3.2 and 4.1 have no common history. Gries [6] recommends use of indentation to indicate the relationship between various tasks described in the step-wise refinement process. However, on a medium with a limited extent in both horizontal and vertical directions (such as the PLATO IV plasma panel), the use of task names indicates task relationships just as clearly, while simplifying the screen management problem.

Conclusion

It is generally agreed that introductory programming courses are not sufficiently concerned with teaching problem-solving methods and structured programming concepts. One way for students to learn these ideas is by following example problem solutions, but there are seldom enough teachers or graders available to provide such personal attention. Under these conditions, a CAI tutorial system which uses an interactive dialogue to guide a student through a top-down solution to a complex programming problem becomes an attractive idea. This paper has discussed one such system.

The heart of the system is an AND-OR graph representing both the student's solution process and the solution program for a complex problem. Student and tutor traverse this graph in the process of solving the problem. There is a great deal of effort involved in developing such a solution graph for a sizable problem. However, such a representation provides several advantages which make the effort worthwhile, namely:

(1) it allows the tutor to effectively engage in a natural language dialogue using a simple parsing scheme.

(2) it enables the system to provide detailed comments on both the student's solution and his problem solving techniques.

(3) it allows such a machine tutor to deal with rather large problems, providing additional emphasis of the need for good program structure and proper problem solving methods.

References

1. Alpert, D. and D. L. Bitzer, "Advances in Computer-based Education", Science 167 (1970), pp 1582-1590

2. Dijkstra, E. W., "Notes on Structured Programming", Technical Report No. 70-WSK-03, Technological University, Eindhoven, The Netherlands, 1970

3. Ells, Tom D. and Peter Freeman, "Design Rationalization of Three BASIC Systems", Technical Report No. 38, Department of Information and Computer Science, University of California, Irvine, November, 1973

4. Fenichel, R. R., J. Weizenbaum and J. C. Yochelson, "A Program to Teach Programming", Comm. ACM, Vol. 13 (1970), pp 141-146

5. Feurzeig, W., P. Wexelblat and R. C. Rosenberg, "SIMON - A Simple Instructional Monitor", IEEE Transactions on Man-Machine Systems, Vol. MMS-11, No. 4, December, 1970, pp 174-180

6. Gries, David, "What Should We Teach in an Introductory Programming Course", SIGCSE Bulletin, Vol. 6, No. 1, February, 1974, pp 81-89

7. Koffman, E. B. and S. E. Blount, "A Modular System for Generative CAI in Machine-Language Programming", Technical Report, Computer Science Group, Electrical Engineering Department, University of Connecticut, December, 1973

8. Nievergelt, Jurg and Edward M. Reingold, "Automating Introductory Computer Science Courses", SIGCSE Bulletin, Vol. 5, No. 1, February, 1973, pp 24-25

9. Tenczar, P. and W. Golden, "Spelling, Word, and Concept Recognition", CERL Report X-35, Computer-based Education Research Laboratory, University of Illinois, October, 1972

10. Ward, Darrell L., "Interactive Directed Programming in Computer Assisted Instruction", unpublished Ph. D. Dissertation, Texas A&M University, August, 1973

11. Wirth, N., "Program Development by Stepwise Refinement", Comm. ACM, Vol. 14 (1971), pp 221-227